KATHOLIEKE UNIVERSITEIT
LEUVEN

Faculty of Economics and Applied Economics

Exact and suboptimal reactive strategies for resource-constrained project scheduling with uncertain resource availabilities

Olivier Lambrechts, Erik Demeulemeester and Willy Herroelen

DEPARTMENT OF DECISION SCIENCES AND INFORMATION MANAGEMENT (KBI)

KBI 0702

Exact and Suboptimal Reactive Strategies for Resource-Constrained Project Scheduling with Uncertain Resource Availabilities*

Olivier Lambrechts Erik Demeulemeester

Willy Herroelen†

Department of Decision Sciences and Information Management

Research Center for Operations Management

Faculty of Economics and Applied Economics

Katholieke Universiteit Leuven (Belgium)

Abstract

In order to cope with the uncertainty inherent in practical project management, proactive and/or reactive strategies can be used. Proactive strategies try to anticipate future disruptions by incorporating slack time or excess resource availability into the schedule, whereas reactive strategies react after a disruption happened and try to revert to a feasible schedule. Traditionally, reactive approaches have focused on obtaining a good schedule with respect to the original objective function or a schedule that deviates as little as possible from the baseline schedule. In this paper, we present various approaches, exact as well as heuristic, for optimizing the latter objective and thus encouraging schedule stability. Furthermore, in contrast to traditional rescheduling algorithms, we present a new heuristic that also takes future uncertainty into account when repairing the schedule. We consider a variant of the resource-constrained project scheduling problem in which the uncertainty is modeled by means of unexpected re-

*This research has been supported by project OT/03/14 of the Research Fund of K.U.Leuven, project G.0109.04 of the Research Programme of the Fund for Scientific Research - Flanders (Belgium) (F.W.O.-Vlaanderen) and project NB/06/06 of the National Bank of Belgium.

†Corresponding author. Tel: +32-16-326970; fax: +32-16-326732; e-mail: *willy.herroelen@econ.kuleuven.be*

source breakdowns. The results of an extensive computational experiment are given to compare the performance of the proposed strategies.

1 Introduction

Construction, IT and many other sectors operate in a project-based environment. Projects consist of a set of activities that are undertaken to achieve an objective conforming to specific requirements including constraints of time, cost and resources (ISO, 1990). Because companies seldom work on only one project at a time and often subcontract activities to third parties, it is of prime importance to have a good plan at one's disposal throughout project execution in order to coordinate the activities of the involved parties. This *baseline schedule* specifies the starting time of each activity and indicates which resources will be required at what time. It thus enables the project manager to coordinate resources over several projects, to make agreements with subcontractors, to evaluate performance, to quote reliable milestone completion times to the client and to provide an overview of future activities to internal and external parties (Aytug et al., 2005).

Traditional scheduling methods have only focussed on deterministic environments in which all information is given in advance and is not subject to change. However, this will seldom be the case in practice. Delays may be caused by bad weather conditions, resource failures, absenteeism, activity duration increases, etc. In case the project does not share resources with other projects, is executed without resorting to third parties and proprietary resources are insensitive to schedule changes, these disruptions will only have an impact on the total project duration. This situation is unfortunately all but realistic. In practice, starting activities later than planned will often translate into higher work-in-process inventory costs, penalties for occupying resources longer than required, penalties for having subcontractors start later than originally agreed, etc.

On the other hand, starting them earlier than originally envisaged is also not always without costs. One example is the use of large prefabricated construction elements on a building site. Having the subcontractor deliver those elements earlier than needed will increase inventory costs such as the rental of additional storage space. Another example is the hard to estimate cost that results from schedule nervousness because employees will be unprepared to execute the task at an earlier time than originally planned.

In general, those types of deviation costs can be aggregated into a weighted instability performance criterion. This objective measures project nervousness as the sum of the weighted deviations between the original baseline schedule S^0 that is constructed before project execution starts and the expected actually realized schedule $E(\mathbf{S})$. The starting times of the activities in $\mathbf{S}$ will depend on the originally planned starting times as described by S^0, on the disturbances encountered throughout project execution and on the reactive strategy that is used to restore feasibility, when for example resource breakdowns occur.

The weights in this objective function reflect each activity's flexibility. The flexibility of an activity represents the cost to execute that activity at an earlier or later point in time than was planned in the baseline schedule. For example, activities that are executed by resources that are expensive to rent or by subcontractors, will usually have a higher instability weight than activities executed by standard, company-owned resources. The instability weight of the dummy end activity represents the importance of meeting the project due date. Because meeting this due date is usually deemed more important than starting each activity at the planned starting time, the instability weight of the dummy end activity is often far higher than the instability weights of the other activities.

We propose the use of predictive-reactive approaches for solving the problem of resource-constrained project scheduling with the aim of minimizing schedule nervousness due to unexpected resource breakdowns. Predictive approaches indicate how to build a preschedule that meets the temporal relations between activities and respects the resource availabilities as well as the due date set by the project's client. Furthermore, these predictive approaches can try to avoid the occurrence of disruptions during project execution by building in some form of protection turning them into proactive approaches. Unfortunately, in practice it will not be possible to protect the project in such a way that breakdowns can never occur (Davenport and Beck, 2002) because of the prohibitive cost that would result from such an extensive protection. This implies that reactive strategies will always be needed in order to indicate how to revert to a feasible schedule that respects the new constraints created by the disruption and that does not deviate too much from the baseline schedule S^0. *Proactive-reactive project scheduling* thus implies a combination of a proactive strategy for generating a protected baseline schedule and a reactive strategy to resolve the schedule infeasibilities caused by the disturbances that occur during schedule execution.

In this paper, we focus exclusively on the reactive strategy. Vieira et al.

(2003) introduced a framework for rescheduling manufacturing systems that can also be applied to project scheduling. The first dimension is the rescheduling environment, identifying the set of jobs that are to be scheduled. The authors make a distinction between static and dynamic environments. Here, we assume that the set of activities is known in advance, limiting our scope to the static environment. The second dimension is the rescheduling strategy. It determines whether or not a preschedule is generated. Above, we gave a number of reasons why the use of a preschedule is often preferable in practice. We chose to only focus on scheduling with a baseline schedule. The third dimension, the rescheduling policy, specifies when to reschedule. Vieira et al. (2003) mention two extremes: periodic rescheduling and event-driven rescheduling. Periodic rescheduling only reschedules after a certain time period whereas event-driven rescheduling reschedules whenever a given event (such as a machine failure) occurs. Because the latter approach often gives rise to an excessive amount of rescheduling passes, a combination is often used in practice. Hybrid rescheduling reschedules the system periodically and also when major events occur. Our problem forces us to reschedule whenever an infeasibility occurs. However, a hybrid approach will be introduced that uses different rescheduling strategies depending on the severity of the disruption. Finally, rescheduling methods tell us how a schedule can be generated or repaired. The proposed repair strategies are right-shift rescheduling, partial rescheduling and complete regeneration. Right-shift rescheduling postpones each non-finished job by the amount of time needed to solve the conflict and as such restores schedule feasibility. Partial rescheduling only considers the affected jobs and total rescheduling calculates a totally new schedule.

In the next section, we give an overview of research covering rescheduling in machine and project scheduling environments. The problem we want to solve is formally stated in the third section. In the fourth section, the experimental setup that is used to compare the algorithms is described. A number of exact as well as sub-optimal algorithms to solve the rescheduling problem are introduced in section 6. This set of algorithms is extended in section 7 with procedures for incorporating proactivity when repairing a schedule.

2 Literature overview

The rescheduling problem has been extensively studied in a machine scheduling environment. For project scheduling on the other hand, this subject is virtually void.

Wu et al. (1993) introduce heuristics for rescheduling on a single machine after a machine breakdown with the simultaneous objectives of efficiency (i.e. minimizing the makespan) and stability (i.e. minimizing the deviation from the baseline schedule measured in terms of job starting times and job sequence). To solve this bi-criterion problem, local search heuristics are used. These heuristics start with the optimal solution for the minimal makespan problem (using the procedure described in Carlier (1982)) and the minimal deviation schedule (being the right-shift schedule for the sequence deviation criterion and the schedule minimizing the sum of absolute lateness for the starting time deviation criterion). The heuristics generate a set of non-dominated schedules using adjacent or general interchanges.

Abumaizar and Svestka (1997) compare a number of reactive strategies for rescheduling a job shop subject to machine breakdowns. In 'right-shift rescheduling' the current schedule is shifted to the right by the duration of the repair. In 'total rescheduling' the initial schedule is disregarded and the only focus is on the efficiency measure (in their case minimizing the makespan). Finally, 'affected operations rescheduling' means that only operations that are directly or indirectly influenced by the disruption are rescheduled in order to minimize both the increase in makespan and the deviation from the initial schedule by minimizing the starting time deviation and reducing the sequence deviation to zero. The authors conclude that 'affected operations rescheduling' outperforms 'right-shift rescheduling' for the efficiency criterium but does not perform significantly better than 'total rescheduling'. For starting time deviation however, 'affected operations rescheduling' strongly outperforms the other strategies.

Qi et al. (2006) consider rescheduling for machine planning subject to deviation costs between the original and the planned schedule. Only cases in which the shortest processing time (SPT) rule is optimal for the original problem are considered. Machine breakdowns as well as job duration changes are allowed. The new schedule is evaluated using the original objective function as well as the deviation cost from the original schedule. The authors distinguish between post-disruption management and predictive disruption management. In the first case the disruption was not foreseen and one has to react after it happened. In the

second case one knows in advance which job or machine will be disrupted and can consequently accommodate the disruption. Optimal policies are presented for the single machine and parallel machine problems. The authors conclude that rescheduling problems corresponding to a rather easy original problem (e.g. a problem for which the SPT rule turns out to be optimal) are not too difficult to solve. However, they are not able to predict what would happen if the original problem were NP-hard.

Approaches for project scheduling subject to general disruptions were developed by Yu and Qi (2004) and Wang (2004). Yu and Qi (2004) introduce a disruption management approach for repairing disrupted project schedules for a resource-constrained project scheduling problem with multiple possible execution modes per activity. Their objective is to minimize the cost of the new schedule as well as the deviation from the baseline schedule. The authors consider project network, activity, resource and milestone disruptions. Schedule feasibility can be restored by changing the execution mode of an activity, rescheduling an activity or temporarily increasing the resource availability. The objective function is a weighted combination of the performance of the new schedule, the costs of changing execution modes, the costs of additional resource availability as well as earliness and tardiness penalties. A hybrid mixed integer programming/constraint propagation approach is used to solve the rescheduling problem formulated as an ILP.

Wang (2004), on the other hand, considers the project rescheduling problem as a dynamic constraint satisfaction problem. Unexpected events such as shifts of activity starting times, changes in activity durations, resource breakdowns and additions or removals of temporal constraints during project execution are modeled as additions or deletions of constraints. Resource constraints are regarded as soft constraints that can be violated at a cost, whereas the due date constraint is a hard constraint (and the due date thus becomes a deadline). In order to repair the schedule, a metaheuristic is presented for generating a feasible schedule that minimizes the weighted resource constraint violation.

Finally, Van de Vonder et al. (2006a) introduce several heuristic approaches for rescheduling a project subject to activity duration disruptions in order to minimize weighted instability. Four reactive approaches are introduced. First of all, simple priority rules are used in conjunction with a schedule generation scheme. The second approach is to fix resource allocations in advance, effectively eliminating the need to consider resources during schedule execution and reducing the problem to a resource-unconstrained project rescheduling prob-

lem that is solved by right-shifting the affected activities such that a feasible schedule is generated. A third procedure is a sampling approach that considers several alternative solutions (obtained by combining various priority lists with various schedule generation schemes) at each decision time and selects the best amongst those. Time window sampling is a modification of this sampling approach that focuses on the activities planned to start within a certain time window from the rescheduling point in order not to adhere too much importance to activities for which a great deal of uncertainty remains regarding their actual starting times. Finally, a heuristic procedure for solving the resource-constrained project scheduling problem with weighted earliness and tardiness penalties (RCPSP-WET or $m, 1|cpm|early/tardy$ in the notation of Herroelen et al. (2000)) is used for fully rescheduling the project. The results of a computational experiment show that among the priority rule-based procedures, a priority list in which activities are ordered according to non-decreasing earliest baseline starting times performs best. Fixing resource allocations greatly speeds up the rescheduling process at the expense of a substantially worse instability cost. Unsurprisingly, both sampling approaches outperform simple priority list based policies. The best performing approach was the one based on the RCPSP-WET heuristic.

3 Problem statement

The objective of the proactive-reactive project scheduling problem is to minimize schedule nervousness while meeting precedence, resource and due date constraints. This objective is measured by the sum of the weighted deviations between the original baseline schedule S^0 that is constructed before project execution starts and the expected actually realized schedule $E(\mathbf{S})$:

$$\text{minimize} \sum_{i \in N} w_i |E(\mathbf{s}_i) - s_i^0| \tag{3.1}$$

where s_i^0 denotes the planned starting time of activity i in the baseline schedule S^0, $E()$ denotes the expectation operator, $\mathbf{s}_i$ denotes the actually realized starting time of activity i during project execution and the activity weight w_i denotes the per unit starting time disruption cost for activity i.

In section 1 we already stated that the real starting times $\mathbf{S}$ are stochastic variables that depend on the originally planned starting times as described by

S^0, on the disturbances encountered during project execution and on the reactive strategy that is used to restore feasibility. We will now briefly cover these three factors.

In this paper we will not focus on the construction of the baseline schedule S^0 but assume this is given. For an overview of approaches for determining baseline schedules for various variations of the project scheduling problem in a deterministic setting we would like to refer the interested reader to Brucker et al. (1999), Herroelen et al. (1998) and Demeulemeester and Herroelen (2002). The stochastic setting, on the other hand, is extensively covered by Leus (2003), Leus and Herroelen (2004), Van de Vonder et al. (2005) and Van de Vonder et al. (2006b) for the case of uncertain activity durations and by Lambrechts et al. (2007a) and Lambrechts et al. (2007b) if the resources are subject to unexpected breakdowns. This baseline schedule is represented by means of a vector of activity starting times $S^0 = (s_1^0, s_2^0, ..., s_n^0)$ and has to satisfy the precedence constraints, the resource constraints and the due date constraint. The precedence constraints represent temporal relations between the activities constituting the project. We assume zero-lag finish-start precedence relations, implying that if activity j is a direct successor of activity i ($j \in SUCC_i$) then j can only start after activity i, with a duration equal to d_i, is completed, or formally:

$$s_j \geqslant s_i + d_i \qquad \forall (i,j) \in A \text{ with } i,j \in N \tag{3.2}$$

with A the set of arcs representing the precedence relations and N the set of nodes representing the activities in the commonly used graph-based activity-on-the-node representation of a project network ($G = (N, A)$).

We assume R different renewable resource types with a per-period availability of a_k. The resource constraints then imply that for each time period t and for each resource type k the cumulative resource requirements of the activities that are in progress during period t ($i \in \mathcal{S}_t$) cannot exceed these availabilities or:

$$\sum_{i \in \mathcal{S}_t} r_{ik} \leqslant a_k \qquad \forall k, \forall t \tag{3.3}$$

Finally, the due date constraint states that the project has to end before time δ:

$$s_n \leqslant \delta \tag{3.4}$$

with n the dummy end activity (and 1 correspondingly the dummy start activity) having a duration and resource use equal to 0.

The second determinant of $\mathbf{S}$ is the disturbance scenario. We consider the case of unexpected resource disruptions. Each of the R resource types is modeled as a set of a_k resource units that are each subject to breakdowns. In case one of these resource units breaks down, it may well be possible that the project schedule is rendered infeasible because one or more resource constraints are violated. This means that the resource availability of each resource type k during each period t is actually a stochastic variable $\mathbf{a}_{kt} \leqslant a_k$.

In case an infeasibility occurs due to a resource breakdown, schedule feasibility needs to be restored by postponing one or more of the offending activities in progress on the resource type causing the infeasibility during the period the disruption occurs. This brings us to the final factor determining $\mathbf{S}$. Postponing this activity or these activities will of course also have an impact on the remainder of the project and therefore all the affected activities will potentially have to be rescheduled in order to obtain a new schedule that respects all constraints. Our global objective is to minimize schedule instability. In case the encountered disruption is the last disruption until project completion, the optimal policy will be to create a feasible schedule for which the weighted deviation from the preschedule is as small as possible. We call this case problem $P1$ and will extensively study it in section 6. However, in practice, we will usually continue facing resource breakdowns. Analytically determining the impact of a rescheduling decision on the future expected project stability is computationally too demanding. An alternative would be simulation. In practice, however, this will be impractical due to the large number of rescheduling decisions that needs to be taken throughout schedule execution. Therefore, we present a bi-objective approach in section 7 that tries to generate a schedule that is feasible, does not deviate too much from the original baseline schedule and is well protected against the occurrence of future disruptions measured by means of a surrogate robustness metric. This problem is then called $P2$.

Before moving on to the actual solution procedures for $P1$ and $P2$, we will formally describe both problems.

First, however, we need to define the relationship between time points and time periods. The need for this distinction stems from the fact that an activity starts and ends at a specific point in time whereas a resource is available or unavailable during a certain time period. In project scheduling literature it is usually assumed that the project starts at time point 0. A number of integer

time points are then specified over the project time horizon. In this paper, we denote the time period between the starting time of the project ($t = 0$) and the first such time point ($t = 1$) as time period 0. This means that if an activity starts at time t and has a duration of d time units, it is assumed to start execution at the beginning of time period t and to finish at the end of time period $t + d - 1$.

Let $S^{t^*} = s_1^{t^*}, ..., s_n^{t^*}$ be the current schedule for the time period t^* in which the infeasibility occurs. The set C includes the activities that were finished by the beginning of this time period ($C = \{i \in N : s_i^{t^*} + d_i \leqslant t^*\}$), its complement is the set $NC = N \backslash C$. In period t^* we know the realizations of $\mathbf{a}_{kt}$ up to the current period, since for $P1$ the current disruption is assumed to be the last, the future resource availabilities are known and equal to the availability a_k in case no information is assumed to be given regarding the potential duration of a breakdown (this assumption will be dropped in the computational experiment of section 7). The real resource availability vector a'_{kt} can then be written as: $a'_{kt} = a_{kt}$ for $t \leqslant t^*$ and $a'_{kt} = a_k$ for $t > t^*$. Our objective is to find a new, repaired schedule S'^{t^*} that satisfies the adapted resource constraints and is as close as possible to the baseline schedule S^0.

Problem $P1$ can then be written as:

$$\text{minimize} \sum_{i \in NC} w_i |s_i'^{t^*} - s_i^0| \tag{3.5}$$

subject to

$$s_i'^{t^*} + d_i \leqslant s_j'^{t^*} \qquad \forall (i,j) \in A \tag{3.6}$$

$$\sum_{i \in \mathcal{S}_t} r_{ik} \leqslant a'_{kt} \qquad \forall t, \forall k \tag{3.7}$$

$$s_i'^{t^*} = s_i^{t^*} \qquad \forall i \in C \tag{3.8}$$

Problem $P2$ is an extension of problem $P1$. The main difference is that the objective function now tries to minimize the deviations between the expected realized schedule and the baseline schedule instead of the repaired schedule and the baseline schedule. Therefore, the procedures considered in section 7 simultaneously try to minimize the deviation from the baseline schedule caused by the repair action and to maximize the robustness of the repaired schedule. Furthermore, a'_{kt} becomes $\mathbf{a}_{kt}$ for $t > t^*$ instead of a_k.

Table 1: Parameter settings for the 480 test instances

Number of activities (excl dummies)	30
Activity durations	Randomly from [0,10]
Number of resource types	4
Network Complexity	1.5, 1.8 or 2.1
Resource factor	0.25, 0.50, 0.75 or 1.00
Resource strength	0.20, 0.50, 0.70 or 1.00
Instability weights non-dummy	$P(\mathbf{w}_i = x) = 0.21 - 0.02x$
Instability weight dummy-end	$10E(\mathbf{w}_i)$
Mean times to failure	Uniformly from $[s_n^{min}, 2s_n^{min}]$
Mean times to repair	Uniformly from [1,5]

4 Experimental setup

In the following two sections, the reactive approaches for $P1$ and $P2$ will be presented together with computational results from an experiment we set-up in order to be able to assess the efficiency and effectiveness of each approach. For each instance, a baseline schedule is generated according to a certain proactive strategy. This baseline schedule is then executed until a resource disruption occurs or until the project finishes. Each time a resource breakdown causes an infeasibility, this infeasibility is resolved by means of a reactive strategy. Note that we assume that whenever an activity is preempted in order to solve a conflict, it needs to be restarted from scratch (*preempt-repeat assumption*) and that activities can never start before their baseline starting time: $s_i \geqslant s_i^0$ (*railroad scheduling assumption*) unless the opposite is indicated.

We used the test instances contained in the well-known PSPLIB set of project network instances (Kolisch and Sprecher, 1997). Because of the demanding computation time of the exact reactive procedure, we restricted our research to the 480 30-activity networks of PSPLIB. However, there is no reason to believe the same results do not hold for 60, 90 and 120 activity networks. Furthermore, in more computationally demanding cases suboptimal approaches may be used instead of exact ones (see e.g. Van de Vonder et al. (2006a)).

The parameter settings used to generate the instances are shown in Table 1, with s_n^{min} the minimal makespan for the deterministic problem, i.e. the optimal solution of the resource-constrained project scheduling problem.

We combined three proactive baseline scheduling procedures (described in section 5) with the exact and suboptimal procedures described in sections 6 and 7. Each combination of a proactive policy and a reactive policy was tested

using 10 replications for each problem instance, each having different mean times to failure ($MTTF_k$) and mean times to repair ($MTTR_k$). The distribution of these parameters is shown in Table 1. These values then unambiguously define the breakdown and repair process for each resource unit if we assume that repair times and times to breakdown are exponentially distributed (the rationale for this assumption is given in Lambrechts et al. (2007a)). Finally, the project due date is derived from the minimal makespan schedule. In a static and deterministic environment, the lower bound on the makespan corresponds to the makespan of the schedule obtained when optimally solving the RCPSP. It seems reasonable to assume that the project manager will prefer a makespan that does not deviate too much from this lower bound. Therefore, we set the due date of the robust schedule 30% above the minimal makespan of the project.

5 Proactive baseline scheduling procedures

In this section we give a short overview of the various proactive strategies we used for generating a baseline schedule. A more extensive treatment can be found in Lambrechts et al. (2007a). In our proactive baseline generation process three choices need to be made.

First of all, one has to decide whether to start from a minimal makespan schedule that is short but usually also very dense and therefore prone to disruption or, alternatively, from a schedule in which activities with a high impact on total project instability are scheduled as early as possible in time (*'highest cumulative instability weight (CIW) first'*) in order to decrease the probability that these activities get disrupted due to the disruption of an activity earlier in the schedule.

Secondly, it has to be decided whether to apply resource buffering to this initial schedule. Resource buffering boils down to planning the project using a resource availability that is lower than the actual resource availability a_k. Since we assume that uncertainty is modeled by means of resources that are subject to random breakdowns, using less resources per time unit than the maximal availability can prevent the negative impact of these breakdowns.

Finally, time buffering can be added. This implies that we explicitly insert idle time into the schedule based on the estimated size and impact of activity disturbances on the objective function. In the end, this gives us a total of 2^3 different strategies.

6 Reactive procedures

In this section we present a number of procedures for optimally or sub-optimally solving problem $P1$. Recall that the aim of $P1$ is simply to restore schedule feasibility while minimizing the weighted sum of deviations between the new, repaired schedule S'^{t^*} and the initial, baseline schedule S^0. We first introduce an exact algorithm that reduces the problem to the resource-constrained project scheduling problem with weighted earliness-tardiness penalty costs (RCPSP-WET). Because this approach is computationally very intensive, we also introduce a number of priority list-based heuristic procedures that yield an acceptable solution with minimal computational effort. Finally, a strategy is introduced combining optimal and suboptimal procedures in order to yield good solutions within an acceptable time frame.

We introduce the example network in Figure 1 to illustrate the various strategies we present in this paper. This graph represents a project consisting of 10 activities. Above each activity node, we indicate its planned duration, its resource requirement of a single renewable resource type with a deterministic per period availability of 8 units (each subject to breakdowns) and its instability weight. Note that activities 1 and 10 are dummy activities with a duration and a resource usage of 0. Activity 1 indicates the start of the project, whereas activity 10 signals the end. The instability weight for activity 10 is much larger than the other instability weights in order to reflect the fact that in practice meeting the project due date is often deemed more important than meeting planned activity starting times. In this example we assume a project due date of 18. The baseline starting time of the dummy start activity is then set to the release date of the project (time period 0), whereas the dummy end activity is assumed to end at the project due date. Note that for ease of notation and illustration only one resource type is considered, but the examples as well as the algorithms presented in this paper are easily extensible to and will be tested for the multi-resource case.

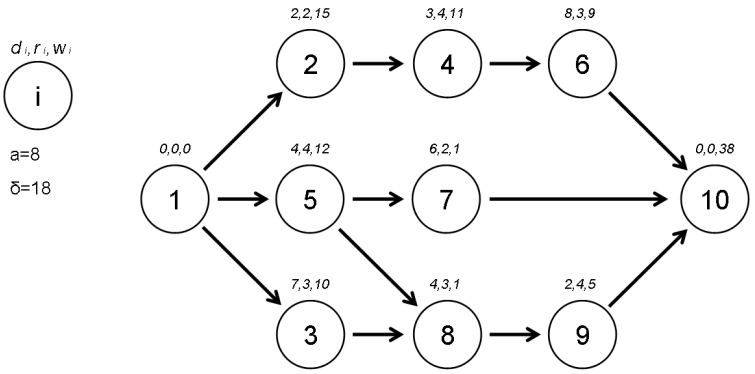

Figure 1: Example project network

6.1 Exact solution procedure

To gain a better understanding of the problem and to be able to quantify the performance gap for the heuristic procedures that we will present in the following subsections, we will now introduce an exact approach for solving the rescheduling problem $P1$.

At first sight, there are quite some resemblances with the well known RCPSP-WET problem. The RCPSP-WET is an extension of the traditional RCPSP (Demeulemeester and Herroelen, 2002). Activities have individual due dates with associated earliness and tardiness penalty costs. Instead of the usual objective of makespan minimization, we now want to find a schedule for which the weighted penalty cost is minimal. Earliness (e_i) and tardiness (t_i) costs are assumed to be linearly related to the number of time units an activity is completed respectively before or after its due date (dd_i).

The main difference between our approach and the RCPSP-WET is that in our approach some activities have fixed starting times (i.e. the activities in C and the activities that are in progress during period t^* but that are not preempted) and the addition of the non-retroactivity constraint (Van de Vonder et al., 2006a). This constraint prevents that activities are scheduled in the past. Consider for example activity i which was supposed to start at s_i^0 but was preempted a number of times. In case we do not add any additional restrictions on the starting time of i, it could possibly be scheduled at $s_i'^{t^*} < t^*$.

An example is shown in Figure 2. The horizontal axis represents time, whereas the vertical one represents resource usage. Two unrelated activities, activity 1 and activity 2 are executed in parallel. The first activity has a duration

of 5 and a resource usage of 3, the second activity has a duration of 4 and a resource usage of 1. Their respective instability weights are 10 and 1.

The baseline schedule is depicted in the top left chart. Both activities are scheduled to start at the beginning of the time period 0. However, as is shown in the second chart, a resource breakdown of one unit occurs during this period forcing the preemption of activity 2 (since this results in the lowest total instability cost) and postponing activity 2 until period 1. A similar situation occurs in periods 1, 2 and 3. In each time period activity 2 is preempted and has to be repeated from scratch. In period 4 however, two units break down. This means that postponing activity 2 is not sufficient for solving the conflict and that we will postpone activity 1. In case no additional constraint is imposed on the starting time of activity 2, this activity will be scheduled again in time period 0. Clearly, this is not possible in practice as one cannot schedule activities in the past. Therefore, we solve a reduced problem when faced with problem $P1$, which only includes the non-completed activities and which starts at time point t^*. Because of this starting time, no activity can be scheduled earlier than the period during which the disruption occurs, meaning that the non-retroactivity constraint will be satisfied.

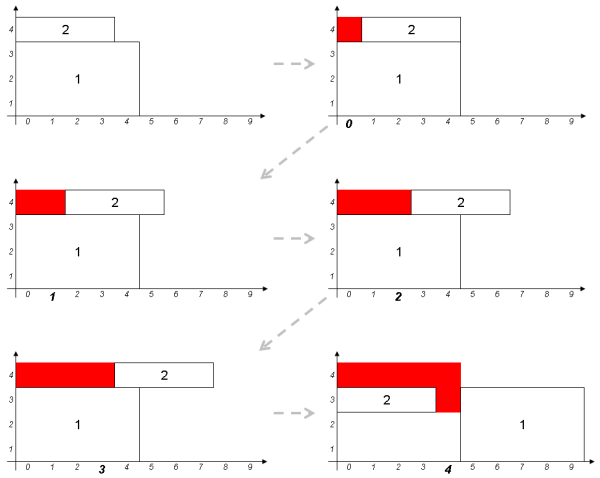

Figure 2: Illustration of the non-retroactivity constraint

When solving $P1$ we need to solve a number of RCPSP-WET instances, namely one for each *preemption alternative*. A preemption alternative is defined as a subset $\mathcal{P}$ of the set of activities in progress during period t^* ($\mathcal{P} \subset \mathcal{S}_{t^*}$)

so that the cumulative resource requirements of the non-preempted activities ($i \in \mathcal{S}_{t^*} \backslash \mathcal{P}$) do not exceed the real availabilities a'_{kt^*} in period t^*. We only need to consider *minimal preemption alternatives*, i.e. preemption alternatives for which it is impossible to remove a single activity from the subset and still retain a preemption alternative satisfying the adapted resource constraints. It can easily be shown that non-minimal preemption alternatives can never lead to a better solution value because instability weights are assumed to be non-negative. A full enumeration scheme is used to determine the set of all minimal preemption alternatives Ψ as can be seen in line 5 of Algorithm 1.

In our algorithm, $S'^{t^*}_{best}$ represents the best repaired schedule over all preemption alternatives and $\mathcal{I}(S'^{t^*}_{best})$ the corresponding objective function value.

When solving problem $P1$, $|\Psi|$ reduced RCPSP-WET instances will be solved. Each of those reduced instances considers a scheduling problem from the time of disruption onwards. Therefore, the completed activities ($i \in C$) are removed from the network (line 9-11). For activities that were busy at t^* but that are not preempted ($i \in S_{t^*} \backslash \mathcal{P}$), no scheduling decision needs to be made as non-preemption implies that $s'^{t^*}_i = s^{t^*}_i$. However, in order to correctly incorporate the constraints this decision imposes on the rest of the problem, we need to represent this in the network. The activities representing the non-preemption decisions have a modified duration that is obtained by subtracting the executed part of their duration from their original durations. Furthermore, these activities are supposed to start at time 0 with an earliness cost and a tardiness cost set to infinity and a due date set to the time corresponding to their original finishing time $s^{t^*}_i + d_i - t^*$ (line 14-18). This means that we only need to consider busy but preempted as well as not-yet-started activities for rescheduling. Both types will incur tardiness penalty costs that are derived from their corresponding instability weights. In case the railroad scheduling assumption is imposed, all earliness costs will be set equal to infinity. If not, the instability weight is used. The activity due dates are derived from the baseline schedule ($dd_i = s^0_i + d_i - t^*$) or set to 0 if dd_i would become negative otherwise (line 20-22). Arcs are removed as needed and inserted between the dummy start activity and those activities that have no predecessors any longer because of activity removal for generating the reduced network. Finally, a dummy activity x is included that represents the reduced resource availability due to the breakdown causing the infeasibility. This activity has a resource usage equal to the number of broken down resource units, a duration equal to the duration of the resource disruption (which we assume to be 1 for the moment as indicated above) and its starting time is fixed

Algorithm 1 Exact algorithm for P1

$t^* :=$ time period for which $\exists k : \sum_{i \in \mathcal{S}_{t^*}} r_{ik} > a'_{kt^*}$
$C := \{i \in N : s_i^{t^*} + d_i \leqslant t^*\}$
$NC := N \backslash C$
$\mathcal{I}(S'^{t^*}_{best}) := \infty$
$\Psi := \{\mathcal{P} : \sum_{i \in \mathcal{S}_{t^*} \backslash \mathcal{P}} r_{ik} \leqslant a'_{kt^*}$ AND $\nexists j \in \mathcal{P} : \sum_{i \in \mathcal{S}_{t^*} \backslash \{\mathcal{P} \backslash \{j\}\}} r_{ik} \leqslant a'_{kt^*}\}$
while $\Psi \neq \emptyset$ **do**
  pick $\mathcal{P}$ from Ψ and $\Psi := \Psi \backslash \mathcal{P}$
  $G' = (N', A') := G = (N, A)$
  for $i := 2$ to n **do**
    if $i \in C$ **then**
      remove i from G'
  $dd_1 := 0$, $e_1 := \infty$, $t_1 := \infty$
  for $i \in N' \backslash \{1, n\}$ **do**
    if $i \in \mathcal{S}_{t^*} \backslash \mathcal{P}$ **then**
      $d_i := s_i^{t^*} + d_i - t^*$
      $dd_i := d_i$
      $e_i := \infty$
      $t_i := \infty$
    else
      $dd_i := max(0, s_i^0 + d_i - t^*)$
      $e_i := \infty$ OR $e_i := w_i$
      $t_i := w_i$
  $dd_n := max(0, s_n^0 - t^*)$, $e_n := \infty$, $t_n := w_n$
  add dummy activity x to N': $d_x :=$ duration breakdown, $r_{xk} := a_k - a'_{kt^*}$
  $A' := A' \cup \{(1, x), (x, n)\}$
  $dd_x := d_x$, $e_x := \infty$, $t_x := \infty$
  solve RCPSP-WET yielding schedule S^{WET}
  for $i := 1$ to n **do**
    if $i \in C$ OR $i \in \mathcal{S}_{t^*} \backslash \mathcal{P}$ **then**
      $s_i'^{t^*} := s_i^{t^*}$
    else
      $s_i'^{t^*} := s_i^{WET} + t^*$
  if $\mathcal{I}(S'^{t^*}) < \mathcal{I}(S'^{t^*}_{best})$ **then**
    $\mathcal{I}(S'^{t^*}_{best}) := \mathcal{I}(S'^{t^*})$
    $S'^{t^*}_{best} := S'^{t^*}$
$S'^{t^*} := S'^{t^*}_{best}$

at time 0 in a similar way to that of the busy but non-preempted activities (line 24-26).

We will illustrate this approach on the disrupted schedule for our example network that is shown in Figure 3.

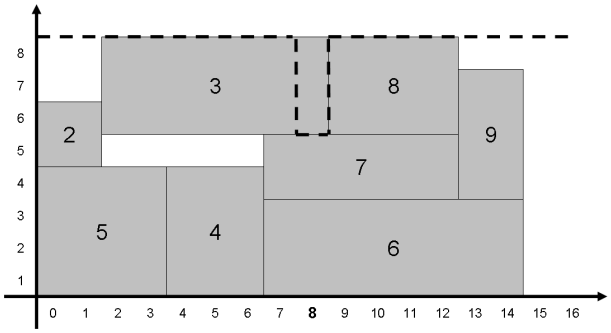

Figure 3: Minimal makespan schedule disrupted by the breakdown of 3 resource units in period 8

The schedule is disrupted because of the one-period breakdown of 3 resource units in period $t^* = 8$ resulting in a violation of the resource constraints: $\sum_{i \in \mathcal{S}_8} r_i = 8 > a'_8 = 5$. The preemption alternatives are then: (3), (6), $(3, 6)$, $(3, 7)$, $(6, 7)$ and $(3, 6, 7)$, of which clearly only (3) and (6) are minimal. Let us consider the reduced network corresponding to preemption alternative (3) shown in Figure 4. Above each activity we indicate its parameters as follows: d_i, r_i, e_i, t_i, dd_i.

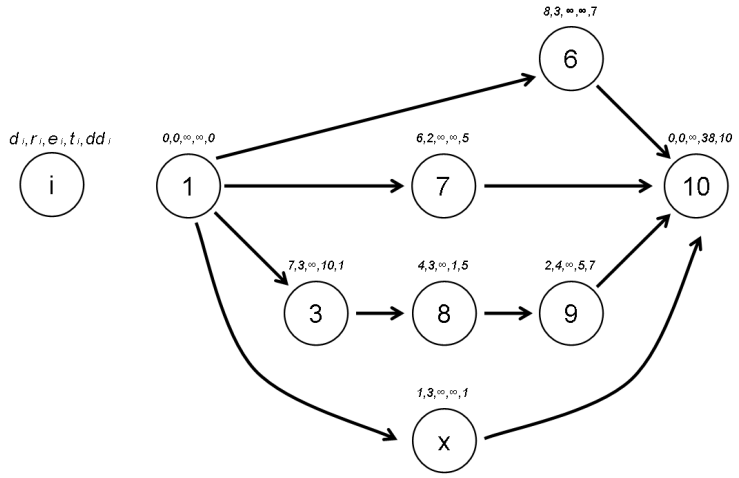

Figure 4: Reduced network for $t^* = 8$ with $\mathcal{P} = (3)$

In order to optimally solve the RCPSP-WET, the exact branch-and-bound algorithm by Vanhoucke et al. (2001) is used. This solution procedure computes lower bounds using an exact recursive search algorithm for the resource-constrained project scheduling problem with weighted earliness and tardiness penalties. Resource conflicts are resolved by adding extra precedence relations based on the concept of minimal delaying alternatives (Demeulemeester and Herroelen (1992) and Demeulemeester and Herroelen (1997)).

For our example, the results of optimally solving the reduced network instances corresponding to both minimal preemption alternatives are shown in Figure 5. The top schedule corresponds to preemption alternative (3) and has an objective function value equal to 264, whereas the bottom schedule corresponds to alternative (6) and performs far better with an objective function equal to 18.

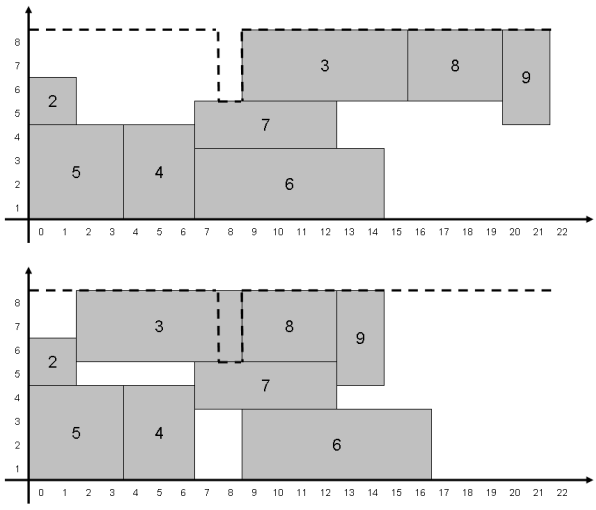

Figure 5: Optimally repaired schedule for $\mathcal{P} = (3)$ (above) and $\mathcal{P} = (6)$ (below)

Note that it is possible that this procedure requires a prohibitively long computation time. In that case the branch-and-bound approach is truncated by terminating it after a chosen period of time, yielding the best solution that was found so far. Results from Vanhoucke et al. (2001) show that for their test setting all problems with 10 activities could be solved to optimality within 1 second of CPU time on a Pentium III, 800 Mhz processor. This was the case for 91% of the 20 activity-problems and for 85.9% of the 30 activity-problems. Because the project parameters are quite comparable with those of our problem setting, because we use a faster machine and because in our setting usually only a reduced problem ($i \in NC$) needs to be tackled, we chose to terminate the procedure after 0.5, 1 or 2 seconds of CPU time. The results are shown in Table 2. For each proactive policy type we show the average weighted instability objective function value taken over all repetitions over all project network instances for the three different cut-off time values. The average objective function value per reactive strategy over all proactive policies is indicated in italics at the end of each row.

Note that for each combination of a proactive policy, a project network instance and a disruption scenario, the RCPSP-WET procedure is invoked a number of times (once for each preemption alternative at each disruption time). The procedure to solve $P1$ is thus used as a myopic procedure that tries to restore feasibility with minimal deviation from the original schedule, while ig-

Table 2: Experimental results for the exact procedure

	no time buffering				time buffering				
	no res buffering		res buffering		no res buffering		res buffering		
	min Cmax	max CIW	min Cmax	max CIW	min Cmax	max CIW	min Cmax	max CIW	
Optimal (0.5s)	163.17	123.09	61.50	63.16	97.79	86.44	44.46	51.84	*86.43*
Optimal (1s)	161.93	121.21	60.86	62.72	97.22	85.90	44.37	51.50	*85.71*
Optimal (2s)	160.84	120.53	60.68	62.86	97.01	85.62	44.31	51.08	*85.37*

Table 3: Percentage of times the reduced problem is solved to optimality

Optimal (0.5 s)	92.5 %
Optimal (1 s)	93.9 %
Optimal (2 s)	94.9 %

noring the potential occurrence of future disruptions. This means that it is only optimal as long as no more disruptions occur, but might well be suboptimal in case this assumption is dropped. How well it performs compared with a reactive procedure that tries to protect against future disruptions will be shown in section 7.

As expected, the procedure performs better, the longer it is allowed to run before being truncated. However, the difference between 0.5 s and 1 s is more pronounced than the one between 1 s and 2 s. In Table 3 we show the fraction of the number of times that the branch-and-bound approach for solving RCPSP-WET was invoked and that it actually found the optimal solution within the allotted time. From both tables we can conclude that truncating the procedure after 1 second seems reasonable.

Furthermore, in Table 4 the results are shown for the exact procedure truncated after 1 second with and without the railroad scheduling assumption.

Perhaps surprisingly, dropping the railroad scheduling assumption actually yields worse results in some cases. The reason might be that although a better solution can be found for problem $P1$, this does not necessarily hold when considering iterative rescheduling along the project horizon because built-in protection is partially lost by scheduling activities earlier than originally planned. Nevertheless, for three proactive strategies (of which two are actually the best performing ones overall) allowing activities to be scheduled earlier than their

Table 4: Optimal procedure with or without railroad scheduling

	no time buffering				**time buffering**				
	no res buffering		**res buffering**		**no res buffering**		**res buffering**		
	min Cmax	**max CIW**	**min Cmax**	**max CIW**	**min Cmax**	**max CIW**	**min Cmax**	**max CIW**	
Optimal (1s, RR)	161.93	121.21	60.86	62.72	97.22	85.90	44.37	51.50	*85.71*
Optimal (1s, no RR)	171.40	124.85	62.37	62.03	99.22	87.09	44.19	50.34	*87.69*

Table 5: Computation times for the optimal procedure

Optimal (0.5s)	0.088 s
Optimal (1s)	0.141 s
Optimal (2s)	0.222 s
Optimal (1s and no railroad)	0.191 s

baseline starting time proved to be slightly better.

The average computation times over all project instances for all the procedures we presented in this section are shown in Table 5.

6.2 Heuristic procedures

Clearly, optimally solving problem $P1$ will not always be feasible in practice. The computational requirements of the exact branch-and-bound approach render it unsuitable for large project networks. In order to overcome this problem, we present a number of fast solution procedures that are able to generate a repaired schedule with a reasonable instability cost in a short amount of time.

6.2.1 List scheduling

Inspired by the promising results of the use of priority lists in machine and project scheduling, we propose to use a simple reactive strategy relying on *list scheduling*. First of all, a *random* precedence feasible priority list is included for benchmarking purposes. However, we expect far better results from a *scheduled order list* that allows us to reschedule the activities in the order dictated by the baseline schedule (the lowest activity number being the tie-breaker), while taking into account the new, reduced resource availabilities. More specifically, when a disruption occurs in period t^*, we create a priority list L including the

activities that are not yet completed at t^* ($i \in NC$), ordered in non-decreasing order of their baseline starting times s_i^0.

This priority list is then decoded into a feasible schedule S'^{t^*} using a *modified serial schedule generation scheme* that takes the known resource availabilities a'_{kt} up to the current time period t^* into account. The modification of the serial schedule generation scheme has to do with the case where the current activity taken from the list is in progress but not yet completed when the infeasibility occurs. This activity can be left unchanged, or it can be interrupted and repeated (recall that we assumed a preempt-repeat setting). The pseudocode for this procedure is given in Algorithm 2. The new activity starting times are denoted as $s_i'^{t^*}$ and the set of direct predecessors of activity i as $PRED_i$. Note that in Algorithm 2 it is required that the list L is precedence feasible. This means that an activity is never allowed to be included in L in front of one of its predecessors. This condition will automatically be satisfied for the scheduled order list because precedence feasibility of a schedule implies precedence feasibility of the corresponding starting time based ordering of the activities.

Algorithm 2 Modified Serial Schedule Generation Scheme

$t^* :=$ time period for which $\exists k : \sum_{i \in \mathcal{S}_{t^*}} r_{ik} > a_{kt^*}$
$C := \{i \in N : s_i^{t^*} + d_i \leqslant t^*\}$
$NC := N \backslash C$
$L :=$ precedence feasible ordered list with activities $i \in NC$
for $i := 1$ to n **do**
  if $i \in C$ **then** $s_i'^{t^*} := s_i^{t^*}$
for $p := 1$ to $|L|$ **do**
  if $s_{L_{(p)}}^{t^*} < t^*$ **then** $s_{L_{(p)}}'^{t^*} := s_{L_{(p)}}^{t^*}$
  else $s_{L_{(p)}}'^{t^*} := max\{s_{L_{(p)}}^0, t^*, max_{i \in PRED_{L_{(p)}}}(s_i'^{t^*} + d_i)\}$
  while $\exists k, t : \sum_{i:i \in \mathcal{S}_t} r_{ik} > a'_{kt}$ **do**
    if $s_{L_{(p)}}'^{t^*} = s_{L_{(p)}}^{t^*}$ **then** $s_{L_{(p)}}'^{t^*} := t^* + 1$
    else $s_{L_{(p)}}'^{t^*} := t + 1$

Activities selected from the list are scheduled as early as possible. For activities that are in execution during the time of disruption t^*, this means that the procedure first tries the current scheduled starting time $s_i^{t^*}$. If this turns out to be infeasible, the procedure searches for feasibility by starting the activity in the next time period (t^*+1) and subsequent time periods if necessary. For activities that did not start yet, it is only necessary to consider the earliest precedence

feasible starting time (or the current time period t^* if the latter turns out to be larger). Note that, as we stated before, we never allow an activity to start before its baseline starting time s_i^0.

6.2.2 Tabu search based improvement heuristic

The *scheduled order list* approach is able to very quickly generate feasible solutions with a reasonable quality. However, solutions may be improved by superimposing a tabu search based improvement heuristic (Glover and Laguna, 1993) on the priority list rule. This procedure will try to improve the starting solution by iteratively executing the best precedence feasible interchange of two activities in the priority list that does not lead to a state included in the tabu list. The objective is to find a precedence feasible ordering of activities corresponding to a feasible schedule that deviates as little as possible from the baseline schedule S^0. The advantage of tabu search is that by using a tabu list (a list of moves or states that are forbidden for a number of iterations) the procedure can also choose non-improving moves so that it avoids getting stuck in local optima like traditional local search approaches. The procedure is explained in Algorithm 3.

Our implementation considers a maximum number of iterations ($MAXITER$) that has to be executed before the procedure ends and includes a frequency based penalty function to further prevent cycling. The length of the tabu list is set to $|L|$. The best solution that is found so far is stored in L_{best} and the corresponding schedule $S'^{t^*}_{best}$ has an objective function value equal to $\mathcal{I}(S'^{t^*}_{best})$. $\mathcal{I}_{temp}$ then is the objective function value of the best (adjacent) interchange found so far in the current iteration. The frequency based penalties are stored per pair (i, s_i) in the variables $\mathcal{F}_{i,s_i}$. Likewise, the tabu status is stored in the variables $\mathcal{T}_{i,s_i}$. Observe that an aspiration criterion is included: in case a tabu solution L has an objective function that outperforms the best solution value that was found so far ($\mathcal{I}(S'^{t^*}) < \mathcal{I}(S'^{t^*}_{best})$), then the tabu status will be overridden and the solution will be stored anyway.

6.2.3 Results

If we apply the scheduled order procedure to the disrupted schedule that was shown in Figure 3, we obtain the priority list $L = (3, 6, 7, 8, 9, 10)$. Feeding this priority list into the modified serial schedule generation scheme yields the same schedule as the one obtained when using the exact procedure, shown as the bottom schedule of Figure 5.

Algorithm 3 Tabu search based reactive procedure

$t^* :=$ time period for which $\exists k : \sum_{i \in \mathcal{S}_{t^*}} r_{ik} > a'_{kt^*}$
$C := \{i \in N : s_i + d_i \leqslant t^*\}$
$NC := N \backslash C$
set $L_{best} := L$, $\mathcal{I}(S'^{t^*}_{best}) := \mathcal{I}$(current solution) , $T := |L|$, $iter := 0$
while (iter $<$ MAXITER) **do**
  $\mathcal{I}_{temp} := \infty$, $i^* := 0$, $j^* := 0$
  for $i := 1$ to $|L| - 1$ **do**
    for $j := i + 1$ to $i + 1$ (adj interchanges) OR $|L|$ (all interchanges) **do**
      if feasible swap **then**
        exchange $L_{(i)}$ and $L_{(j)}$
        generate S'^{t^*} by applying the modified SSGS to L
        if $\mathcal{I}(S'^{t^*}) + \mathcal{F}_{L_{(i)}, s'^{t^*}_{L_{(i)}}} + \mathcal{F}_{L_{(j)}, s'^{t^*}_{L_{(j)}}} < \mathcal{I}_{temp}$ **then**
          if $(iter > \mathcal{T}_{L_{(i)}, s'^{t^*}_{L_{(i)}}}$ AND $iter > \mathcal{T}_{L_{(j)}, s'^{t^*}_{L_{(j)}}})$ OR $\mathcal{I}(S'^{t^*}) < \mathcal{I}(S'^{t^*}_{best})$ **then**
            store $i \rightarrow i^*$, $j \rightarrow j^*$
            $\mathcal{I}_{temp} := \mathcal{I}(S'^{t^*})$
        exchange $L_{(i)}$ and $L_{(j)}$
  if $i^* \neq 0$ **then**
    $\mathcal{F}_{L_{(i^*)}, s'^{t^*}_{L_{(i^*)}}} := \mathcal{F}_{L_{(i^*)}, s'^{t^*}_{L_{(i^*)}}} + 1$, $\mathcal{F}_{L_{(j^*)}, s'^{t^*}_{L_{(j^*)}}} := \mathcal{F}_{L_{(j^*)}, s'^{t^*}_{L_{(j^*)}}} + 1$
    $\mathcal{T}_{L_{(i^*)}, s'^{t^*}_{L_{(i^*)}}} := iter + T$, $\mathcal{T}_{L_{(j^*)}, s'^{t^*}_{L_{(j^*)}}} := iter + T$
    generate S'^{t^*} by applying the modified SSGS to L
    if $\mathcal{I}(S'^{t^*}) < \mathcal{I}(S'^{t^*}_{best})$ **then**
      $\mathcal{I}(S'^{t^*}_{best}) := \mathcal{I}(S'^{t^*})$ AND $L_{best} := L$
  $iter := iter + 1$
generate S'^{t^*} by applying the modified SSGS to L

Table 6: Experimental results for the list scheduling based heuristics

	no time buffering				**time buffering**				
	no res buffering		**res buffering**		**no res buffering**		**res buffering**		
	min Cmax	**max CIW**	**min Cmax**	**max CIW**	**min Cmax**	**max CIW**	**min Cmax**	**max CIW**	
random	655.00	594.21	243.33	249.90	479.14	456.68	192.37	226.78	*387.18*
scheduled order	212.53	168.82	73.75	77.71	124.58	114.05	52.28	64.43	*111.02*
TS - adj - 50 reps	168.91	127.13	63.01	64.32	100.68	90.68	46.30	53.67	*89.34*
TS - adj - 100 reps	167.17	126.39	62.59	63.18	99.66	89.69	46.27	53.08	*88.50*
TS - adj - 200 reps	165.65	125.61	62.35	63.13	99.63	88.96	46.12	53.06	*88.06*
TS - all - 50 reps	157.59	120.13	60.52	62.00	96.06	86.37	44.73	52.03	*84.93*
TS - all - 100 reps	157.07	119.67	60.58	61.99	95.86	85.78	44.84	52.09	*84.74*
TS - all - 200 reps	156.53	119.48	60.62	61.73	95.73	85.93	44.88	52.02	*84.62*

Table 7: Computation times for the list scheduling based heuristics

random	0.000 s
scheduled order	0.000 s
TS - adj - 50 reps	0.015 s
TS - adj - 100 reps	0.022 s
TS - adj - 200 reps	0.035 s
TS - all - 50 reps	0.026 s
TS - all - 100 reps	0.044 s
TS - all - 200 reps	0.078 s

The results of the procedures based on a random priority list, a scheduled order priority list and a scheduled order priority list that is improved by means of tabu search are shown in Table 6. For the tabu search procedure a distinction is made between the results obtained when the procedure is terminated after 50, 100 or 200 iterations and whether only adjacent interchanges or all interchanges are considered. The corresponding computation times per procedure call are shown in Table 7.

Basic list scheduling policies are very fast. All strategies perform significantly better than the random list strategy. It can be observed that scheduled order list scheduling performs quite well given the time necessary to execute the algorithm. However, even when only allowing for adjacent interchanges and 50 iterations, tabu search outperforms random as well as scheduled order list scheduling. These results can even be improved by allowing for general interchanges and more iterations. Surprisingly, when allowing for 200 iterations, the tabu search procedure even sometimes outperforms the exact approach. The reason for this is twofold. First of all, the exact approach is truncated for some

Table 8: Experimental results for the list scheduling based heuristics when dropping the railroad scheduling assumption

	no time buffering				**time buffering**				
	no res buffering		**res buffering**		**no res buffering**		**res buffering**		
	min Cmax	**max CIW**	**min Cmax**	**max CIW**	**min Cmax**	**max CIW**	**min Cmax**	**max CIW**	
scheduled order	211.28	167.35	73.35	77.00	123.82	113.21	51.96	63.82	*110.23*
TS - all - 200 reps	155.44	118.16	60.51	60.95	94.29	84.76	44.64	51.36	*83.76*

instances, potentially leading to a far worse solution than the one that could be found by tabu search in a comparable timespan. Secondly, optimally solving problem $P1$ is no guarantee for obtaining the lowest instability value after a number of iterative rescheduling passes for the same project because of multiple disruptions.

It is interesting to evaluate the performance impact of dropping the railroad scheduling assumption. We modify the adapted serial schedule generation scheme in such a way that it tries to schedule each activity in L as close as possible to its baseline starting time (see also Van de Vonder et al. (2006a)). The difference with the procedure described in Algorithm 2 resides in the scheduling of the activity in case the starting time obtained in lines 8-9 is not feasible. In this case, the approach we use here will first try to schedule the activity one time period earlier if feasible, if this also turns out to infeasible, one period later will be tried. The procedure continues to iteratively try to schedule the activity respectively $x = 1, 2, 3, 4, ...$ periods earlier or later than its baseline scheduling time until a feasible solution is found. The results are shown in Table 8. We observe that the results hardly change. A slight improvement is obtained for most cases at the expense of a slightly larger computation time for the tabu search procedure (0.091 s versus 0.078 s).

6.3 Hybrid procedure

In the previous two sections we presented computationally intensive but exact solution procedures as well as fast but heuristic algorithms for solving problem $P1$. In practice, a project manager will spend less time and effort on small disruptions than on disturbances having a major impact on project stability. Therefore, we present a new approach combining elements from both the ex-

Table 9: Experimental results for the hybrid procedure

	no time buffering				**time buffering**				
	no res buffering		**res buffering**		**no res buffering**		**res buffering**		
	min Cmax	**max CIW**	**min Cmax**	**max CIW**	**min Cmax**	**max CIW**	**min Cmax**	**max CIW**	
Hybrid (10%)	164.34	124.81	60.52	62.66	98.90	87.17	44.02	51.37	*86.72*
Hybrid (25%)	174.41	129.71	61.70	64.09	103.21	91.03	45.25	53.34	*90.34*
Hybrid (50%)	185.89	134.24	63.42	65.57	111.03	95.29	46.13	55.30	*94.61*
Hybrid (100%)	193.26	143.82	66.54	68.08	117.14	100.80	48.03	58.63	*99.54*
Hybrid (200%)	203.16	152.65	69.10	69.93	123.36	107.90	49.25	60.50	*104.48*

act and the heuristic solution procedures. Whenever an infeasibility occurs, a repaired schedule is quickly generated using the 'scheduled order' list-based heuristic. The weighted instability cost $\mathcal{I}(S'^{t^*})$ of this new, repaired schedule S'^{t^*} is then compared with the instability cost of the previous, but now infeasible schedule S^{t^*}. In case $\frac{\mathcal{I}(S'^{t^*})-\mathcal{I}(S^{t^*})}{\mathcal{I}(S^{t^*})} > \varepsilon$, we repair the schedule using the exact branch-and-bound procedure described in section 6.1. If not, S'^{t^*} is retained. The parameter ε is a user-defined cutoff percentage used to determine when a simple heuristic suffices and when an exact approach is required. The pseudocode for this procedure is shown in Algorithm 4.

Algorithm 4 Hybrid reactive procedure

1: $t^* :=$ time period for which $\exists k : \sum_{i \in \mathcal{S}_{t^*}} r_{ik} > a'_{kt^*}$
2: $C := \{i \in N : s_i + d_i \leqslant t^*\}$
3: $NC := N \backslash C$
4: $L :=$ list with activities $i \in NC$ ordered according to non-decreasing s_i
5: generate S' by applying the modified SSGS to list L
6: **if** $\frac{\mathcal{I}(S')-\mathcal{I}(S)}{\mathcal{I}(S)} > \varepsilon$ **then**
7: determine S' using the exact RCPSP-WET procedure

It can be expected that this solution procedure is able to yield good results in terms of stability and this within acceptable computation times. This hypothesis is tested by means of a computational experiment. We consider the following values for ε: $10\%, 25\%, 50\%, 100\%, 200\%$, we terminate the exact procedure after 1 s and use the settings described in section 4. The results of the computational experiment are given in Table 9, the computation times are shown in Table 10.

As expected, the results lie between those of optimal rescheduling and those

Table 10: Computation times for the hybrid procedure

Hybrid (10%)	0.129 s
Hybrid (25%)	0.111 s
Hybrid (50%)	0.087 s
Hybrid (100%)	0.060 s
Hybrid (200%)	0.039 s

of the scheduled order priority list. With only a small increase in required computation time, the procedure is able to yield significantly better results than the simple list scheduling heuristic. However, tabu search based improvement of the scheduled order heuristic still seems to be the most attractive option.

7 Rescheduling for stability and robustness

The approaches we studied up to now for $P1$ were myopic strategies insofar that they try to optimize the global objective of weighted deviation between the baseline schedule and the finally realized schedule by locally minimizing the difference between the baseline schedule and the repaired schedule. However, minimizing the weighted deviation between the rescheduled and the planned activity starting times is no guarantee for optimizing the global objective. It seems naive to assume that schedule uncertainty ceases to exist after schedule feasibility has been restored. Resources remain subject to breakdowns, rework is still an issue and unexpected activity duration increases may still occur. Therefore, it is worthwhile to develop a rescheduling approach that does not only look backwards in time but also adequately tries to protect the schedule from disruptions that might still occur at some future point in time beyond the current disruption. It might be argued that the predictive schedule that was constructed before the project started execution remains well protected in case no new information becomes available during schedule execution. It is exactly this hypothesis that will be tested in this section.

In this section we present a metaheuristic that optimizes the bi-objective problem of weighted deviation minimization combined with robustness maximization. O'Donovan et al. (1999) define robustness as the ability of a schedule to absorb disruptions without affecting planned external activities while maintaining high shop performance. This robustness can be measured by the ex-post stability measure. Unfortunately, evaluation of this objective is very computationally intensive if done through simulation. Therefore, we use a surrogate

measure based on the expected duration increase of an activity due to resource breakdowns.

We extend the tabu search based procedure of section 6.2 by using a bi-objective criterion instead of the single objective of weighted deviation. Our new objective will be a weighted combination of instability ($\mathcal{I}$) and robustness ($\mathcal{R}$):

$$z = \kappa \mathcal{I}(S) + (1-\kappa)\mathcal{R}(S) \tag{7.1}$$

However, because of differences in order of magnitude of both criteria we chose to use relative improvement measures with respect to the starting solution:

$$z = \kappa \frac{\mathcal{I}(S) - \mathcal{I}(S_0)}{\mathcal{I}(S_0)} + (1-\kappa)\frac{\mathcal{R}(S_0) - \mathcal{R}(S)}{\mathcal{R}(S_0)} \tag{7.2}$$

In our computational experiment we assume exponential times to failure and exponential repair times. This allows us to use theorem 1 to derive the expected duration increase $E[\boldsymbol{\delta}_i]$ for each activity.

Theorem 1. *In a preempt-repeat environment with fixed resource allocations, the expected duration extension due to breakdowns for an activity with duration d_i and resource usage r_{ik} of renewable resource type k for which the time to failure of each resource unit is exponentially distributed with parameter λ_k and the time to repair with parameter μ_k is given by:*

$$\frac{\psi}{1-\psi}\left(\frac{1}{\sum\limits_k \lambda_k r_{ik}} + \sum_k \frac{\lambda_k r_{ik}}{\mu_k \sum\limits_l \lambda_l r_{il}}\right) - d_i \tag{7.3}$$

with $\psi = 1 - e^{-d_i \sum\limits_k \lambda_k r_{ik}}$.

This theorem is proven in Lambrechts et al. (2007c) and its results are used to calculate the following schedule robustness measure:

$$\sum_{i \in NC} \sum_{j \in PRED_i} max(w_i(s_j + d_j + E[\boldsymbol{\delta}_i] - s_i), 0) \tag{7.4}$$

In order to assess the impact of varying the weight attributed to schedule robustness in rescheduling we run a number of simulations with different levels of κ. The results are shown in Table 11.

We see that the procedures always perform worse than a pure instability-based strategy such as the tabu search procedure introduced in section 6.2. This

Table 11: Experimental results for the reactive-proactive procedure

	no time buffering				**time buffering**				
	no res buffering		**res buffering**		**no res buffering**		**res buffering**		
	min Cmax	**max CIW**	**min Cmax**	**max CIW**	**min Cmax**	**max CIW**	**min Cmax**	**max CIW**	
$\kappa = 0.2$	272.91	195.05	89.31	85.04	201.18	151.60	71.45	76.29	*142.85*
$\kappa = 0.4$	209.69	150.77	72.11	70.47	142.33	111.87	53.43	61.06	*108.97*
$\kappa = 0.6$	178.32	131.36	65.77	65.67	114.45	95.84	48.53	55.64	*94.45*
$\kappa = 0.8$	164.39	123.44	62.61	63.13	101.42	88.74	46.04	52.68	*87.81*
$\kappa = 0.95$	159.36	120.22	61.42	61.78	96.44	85.80	45.05	51.72	*85.22*

is no doubt due to the fact that we only penalize deviation from the starting schedule. The only thing that matters in our objective function is the finally obtained schedule because this fully determines the objective function value. If the baseline schedule was well protected, the realized schedule usually deviates little from this baseline schedule. However, if it was not well protected, inserting protection during the rescheduling pass only allows us to decrease the number of rescheduling actions, but not to reduce the final instability function value.

For illustration purposes, we also tried to determine the impact of incorporating an estimate of the breakdown duration in the procedure. If a number of resource units are down in period t^*, we assume they remain down up to period $t^* + \mu_k - 1$. The results when we accept this assumption are shown in Table 12. The corresponding computation times are shown in Table 13. Unsurprisingly, the instability performance is far worse.

Things change considerably, however, if we do not only consider instability performance but also penalize the number of rescheduling actions. It does not seem unreasonable to assume that whenever a rescheduling action needs to be performed, this will give rise to extra costs. The result will be that the attractiveness of incorporating proactivity in a reactive strategy will depend on the ratio of rescheduling versus instability costs. The average number of rescheduling passes per project instance is shown in Table 14. We see that rescheduling for robustness reduces the number of rescheduling actions required per instance. The same holds for incorporating extra information regarding breakdown durations.

Table 12: Experimental results for the reactive-proactive procedure taking expected breakdown durations into account

	no time buffering				**time buffering**				
	no res buffering		**res buffering**		**no res buffering**		**res buffering**		
	min Cmax	**max CIW**	**min Cmax**	**max CIW**	**min Cmax**	**max CIW**	**min Cmax**	**max CIW**	
$\kappa = 0.2$	283.97	209.28	109.99	100.13	210.99	167.44	88.60	89.71	*157.51*
$\kappa = 0.4$	227.48	167.36	89.51	86.97	157.22	128.49	71.10	75.59	*125.46*
$\kappa = 0.6$	197.99	148.55	81.70	79.75	129.32	110.30	62.41	69.46	*109.94*
$\kappa = 0.8$	180.51	138.37	78.40	76.55	115.83	102.64	58.96	65.55	*102.10*
$\kappa = 0.95$	175.34	136.48	76.78	76.16	109.94	99.70	58.46	65.35	*99.78*

Table 13: Computation times for the reactive-proactive procedure

	no breakdown estimate	breakdown estimate
$\kappa = 0.2$	0.070 s	0.074 s
$\kappa = 0.4$	0.071 s	0.074 s
$\kappa = 0.6$	0.070 s	0.074 s
$\kappa = 0.8$	0.070 s	0.074 s
$\kappa = 0.95$	0.069 s	0.074 s

Table 14: Average number of times the reactive procedure was invoked per project instance

	no breakdown estimate	breakdown estimate
$\kappa = 0.2$	3.57	1.75
$\kappa = 0.4$	3.65	1.79
$\kappa = 0.6$	3.71	1.81
$\kappa = 0.8$	3.75	1.82
$\kappa = 0.95$	3.76	1.83

8 Conclusion

In this paper we gave an extensive overview of various reactive strategies that can be used during project execution when the project is subject to disruptions due to unforeseen resource breakdowns. We saw that the cost of those breakdowns can be rather high due to the propagation of the resulting disruptions throughout the schedule. Unfortunately, totally eliminating their occurrence is economically unviable. Project managers are therefore forced to accept them as an integral part of project management in practice. In order to reduce the costs these breakdowns generate, one can resort to techniques taking this uncertainty into account while building the project schedule and to good rescheduling techniques enabling the project manager to restore feasibility while incurring an instability penalty that is as small as possible.

In this paper, we exclusively focused on reactive strategies. On the one hand, simple rescheduling heuristics based on a modified serial schedule generation scheme combined with a priority list ordered according to non-decreasing baseline starting times were able to generate reasonably repaired schedules with minimal computational effort. The alternative is to use a dedicated optimal algorithm for solving the RCPSP-WET, effectively generating a feasible schedule that is as close as possible to the original schedule. The drawback of the latter approach is the computational effort required to calculate this schedule. This computational effort can be reduced by using a tabu search heuristic to improve the schedule obtained when using the scheduled order heuristic. A hybrid algorithm combines the extremes of optimal but slow scheduling and sub-optimal but fast scheduling by specifying a cutoff instability cost increase for deciding whether or not an optimal algorithm should be used. Finally, we introduced a metaheuristic that generates schedules that are close to the baseline schedule as well as robust. The attractiveness of those procedures strongly depend on the cost of rescheduling actions.

References

Abumaizar, R. and Svestka, J. (1997). Rescheduling job shops under random disruptions. *International Journal of Production Research*, 35(7), pp 2065–2082.

Aytug, H., Lawley, M., McKay, K., Mohan, S. and Uzoy, R. (2005). Executing

production schedules in the face of uncertainties: A review and some future directions. *European Journal of Operational Research*, 161, pp 86–110.

Brucker, P., Drexl, A., Möhring, R., Neumann, K. and Pesch, E. (1999). Resource-constrained project scheduling: Notation, classification, models and methods. *European Journal of Operational Research*, 112, pp 3–41.

Carlier, J. (1982). The one-machine sequencing problem. *European Journal of Operational Research*, 11, pp 42–47.

Davenport, A. and Beck, J. (2002). A survey of techniques for scheduling with uncertainty. *available from http://tidel.mie.utoronto.ca/publications.php*.

Demeulemeester, E. and Herroelen, W. (1992). A branch-and-bound procedure for the multiple resource-constrained project scheduling problem. *Management Science*, 38, pp 1803–1818.

Demeulemeester, E. and Herroelen, W. (1997). New benchmark results for the resource-constrained project scheduling problem. *Management Science*, 43, pp 1485–1492.

Demeulemeester, E. and Herroelen, W. (2002). *Project scheduling - A research handbook*. Vol. 49 of *International Series in Operations Research & Management Science*. Kluwer Academic Publishers, Boston.

Glover, F. and Laguna, M. (1993). *Tabu Search*. Blackwell Scientific, Oxford. pp 70–141. In C. Reeves (Editor): Modern Heuristic Techniques for Combinatorial Problems.

Herroelen, W., De Reyck, B. and Demeulemeester, E. (1998). Resource-constrained scheduling: A survey of recent developments. *Computers and Operations Research*, 25, pp 279–302.

Herroelen, W., De Reyck, B. and Demeulemeester, E. (2000). On the paper "Resource-constrained project scheduling: Notation, classification, models and methods" by Brucker et al.. *European Journal of Operational Research*, 128(3), pp 221–230.

ISO (1990). *Quality Concepts and Terminology Part One: Generic Terms and Definitions*. International Organization for Standardization.

Kolisch, R. and Sprecher, A. (1997). PSPLIB - A project scheduling library. *European Journal of Operational Research*, 96, pp 205–216.

Lambrechts, O., Demeulemeester, E. and Herroelen, W. (2007a). Proactive and reactive strategies for resource-constrained project scheduling with uncertain resource availabilities. *Journal of Scheduling, to appear*.

Lambrechts, O., Demeulemeester, E. and Herroelen, W. (2007b). A tabu search procedure for developing robust predictive project schedules. *International Journal of Production Economics, to appear*.

Lambrechts, O., Demeulemeester, E. and Herroelen, W. (2007c). Time-slack based techniques for generating robust project schedules subject to resource uncertainty. *unpublished research report*.

Leus, R. (2003). *The generation of stable project plans*. PhD thesis. Department of applied economics, Katholieke Universiteit Leuven, Belgium.

Leus, R. and Herroelen, W. (2004). Stability and resource allocation in project planning. *IIE Transactions*, 36(7), pp 1–16.

O'Donovan, R., Uzsoy, R. and McKay, K. (1999). Predictable scheduling of a single machine with breakdowns and sensitive jobs. *International Journal of Production Research*, 37, pp 4217–4233.

Qi, X., Bard, J. and Yu, G. (2006). Disruption management for machine scheduling: The case of spt schedules. *International Journal of Production Economics*, 103, pp 166–184.

Van de Vonder, S., Ballestin, F., Demeulemeester, E. and Herroelen, W. (2006a). Heuristic procedures for reactive project scheduling. *Computers & Industrial Engineering, to appear*.

Van de Vonder, S., Demeulemeester, E. and Herroelen, W. (2006b). An investigation of efficient and effective predictive-reactive project scheduling procedures. *Journal of scheduling, to appear*.

Van de Vonder, S., Demeulemeester, E., Herroelen, W. and Leus, R. (2005). The use of buffers in project management: The trade-off between stability and makespan. *International Journal of Production Economics*, 97, pp 227–240.

Vanhoucke, M., Demeulemeester, E. and Herroelen, W. (2001). An exact procedure for the resource-constrained weighted earliness-tardiness project scheduling problems. *Annals of Operations Research*, 102, pp 179–196.

Vieira, G., Herrmann, J. and Lin, E. (2003). Rescheduling manufacturing systems: A framework of strategies, policies, and methods. *Journal of Scheduling*, 6(1), pp 39–62.

Wang, J. (2004). Constraint-based schedule repair for product development projects with time-limited constraints. *International Journal of Production Economics*, 95, pp 399–414.

Wu, S., Storer, R. and Chang, P. (1993). One-machine rescheduling heuristics with efficiency and stability as criteria. *Computers and Operations Research*, 20(1), pp 1–14.

Yu, G. and Qi, X. (2004). *Disruption Management - Framework, Models and Applications*. World Scientific, New Jersey.